Seminarbericht
Leichtgewichtige Malwareerkennung in Eingebettetem Code

Benjamin Jochheim

August 31, 2012

Contents

1 Einleitung

Das Ziel dieser Seminararbeit ist die Erstellung einer Thesis-Outline, die die folgende Masterarbeit beschreibt.

1.1 Motivation

Mobile Kommunikationsgeräte, wie Smartphones, PDAs und Tablet PCs sind heutzutage in den Alltag integriert und bieten den Nutzern eine große Palette von Anwendungen.

Viele dieser Anwendungen gehen mit persönlichen Daten um die vor unbefugtem Zugriff geschützt werden müssen.

Die häufiger gewordenen Angriffe auf Mobiltelefone [1] zeigen das ein verbesserter Schutz dringend benötigt wird.

Der Schutz von mobilen Kommunikationsgeräten ist,im Vergleich zu herkömmlichen Desktop PCs, anspruchsvoller. Mobile Kommunikationsgeräte sind in Bezug auf Prozessorleistung, Speicher und Akkugetriebener Stromversorgung gegenüber herkömmlichen PCs benachteiligt.

Durch den mobilen Betrieb benötigen diese Geräte eine Vielzahl von Kommunikationsschnittstellen, die jeweils einen eigenen Software-Stack mitbringen der Angegriffen werden kann. In vielen Kommunikationsszenarien müssen sich die Geräte auf Daten von nicht vertrauenswürdigen Netzen verlassen, um auf zu schützende Daten über das Internet zuzugreifen.

Zum sichern der Kommunikation müssen sich die Geräte auf kryptographie Funktionen verlassen. Dabei haben Angreifer in der Regel wesentlich mehr Rechenleistung zur Verfügung als die mobilen Geräte.

Letztlich sind die Nutzer mobiler Kommunikationsgeräte Endanwender. Sie wollen sich daher auf die Sicherheit ihrer Geräte verlassen können, ohne mit Konfigurationsdetails konfrontiert zu werden.

1.2 Warum werden mobile Geräte angegriffen?

Wie eingangs erwähnt befinden sich wertvolle Daten wie Geschäftskontakte auf den Geräten die schützenswert sind. Ein weiterer Grund für Angriffe auf mobile Geräte sind ihre möglichkeiten indirekt Geld von einem Konto abzubuchen[2]. Kompromittierte Telefone nutzen beispielsweise Premium-SMS Dienste um einer weiteren Person Geld zuzuspielen. Ein weiteres interesanntes Angriffsfeld sind micropayment Anwendungen. Diese werden häufig via NFC (near field communication)[3] durchgeführt. NFC verwendet kurzstrecken funk um Transaktionen durchzuführen, die aufgrund des offenen Funkmediums bereits angegriffen wurden [4].

Mobile Geräte sind vom Internet aus häufig leicht an ihrem markanten IP-Adressbereich der Mobilfunkanbieter erkennbar. Daher sind in diesem Bereich auch remote-Angriffe sehr beliebt.

Wie jeder gewöhnliche Desktop PC benötigen mobile Geräte Softwareupdates um Sicherheitslücken nach ihrem bekanntwerden zu schliessen. Im Gegensatz zu herkömmlichen PCs werden Betriebssysteme (und somit auch die Softwareupdates) speziell für die einzelnen Modelle der mobilen Endgeräte angepasst. Speziell bei dem weit verbreiteten Betriebssystem *Android* kommt es häufig zu der Situation das ein Patch für ein Sicherheitsproblem von Google zur Verfügung steht. Dieser Patch wird jedoch erst wesentlich später von den Geräteherstelern zur Verfügung gestellt. Die immer kürzer werdenden Produktzyklen haben das Problem noch verschärft, indem häufig Geräte im Einsatz sind, die vom Hersteller nicht mehr unterstützt werden. Ein Softwareupdate wird es daher für diese Geräte nicht geben und Sie bleiben damit ungeschützt.

1.3 Schutz vor Angriffen

Um den Schutz mobiler Endgeräte zu veressern wurden verschiedene Maßnahmen vorgeschlagen. Einige Maßnahmen sind bereits weit im Bereich der Desktop Rechner verbreitet. Unter den eingesetzten Verfahren gibt es Signatur basierte Verfahren, Methoden aus dem Bereich *Trusted Computing* und statistische Methoden zur Malwareerkennung.

Signaturbasierte Verfahren nutzen Signaturen bekannter Malware um diese zu erkennen. Der Vorteil dieser Verfahren besteht darin, das sie eine niedrige rate an false-pisitives haben. Es werden also wenig daten fälschlich als Malware erkannt. Nachteile bestehen jedoch darin, das Signaturupdates regelmäßig heruntergeladen werden müssen. Dabei entsteht (eventuell kostenpflichtiger) Traffic. Die Signaturen müssen stets manuell erzeugt werden. Das bedeutet das es immer eine Erkennungslücke zwischen dem auftreten einer Malware und dem

verteilen der Signatur geben wird. Der dritte Nachteil besteht in dem mit wachsender Anzahl an Signaturen steigenden Aufwand.

Die *Trusted Computing* Group schlägt eine Architektur vor [5] , die es ermöglicht nur kryptografisch signierte Anwendungen auf einem Computer starten zu lassen. Anwendungen die nicht signiert sind können nicht zur Ausführung kommen. Die spezifikation wurde für mobile Endgeräte angepasst und als Mobile Trusted Module (MTM) [6] veröffentlicht. Die Architektur benötigt einen Vertrauensanker der in der Lage ist Signaturen zu überprüfen und digitale Schlüssel auf eine sichere Art speichern kann. Damit der Vertrauensanker diese Eigenschaften aufweist, muss er in Hardware bestehen. Zum Einsatz von MTM wird daher spezielle Hardware benötigt die bisher in nur sehr wenigen mobilen Endgeräten verbaut ist. Bei offenen Umgebungen wie Android und seinem Market-Place besteht des weiteren das Problem, das die Software von einer zentralen Stelle signiert werden müsste. Des weiteren kann Trusted Computing nicht bei Angriffen helfen, die Software-Bugs ausnutzt, die in regulär installierter Software vorliegen. Ein Beispiel sind Bugs in einem regulär installierten JavaScript Interpreter.

Statistische Verfahren nutzen statistisch relevante Merkmale um Schadcode zu erkennen. Programmatisch zu erkennen welche Funktion ein ausführbarer Code hat, ist nicht immer möglich*t-cnae-36.* Daher wird das Problem häufig reduziert.

2 Problem Statement

Mobile Endgeräte stehen wie jedes Netzwerkfähige Gerät vor dem Problem, das Sie früher oder später angegriffen werden. Weil mobile Geräte viele Kommunikationsmöglichkeiten besitzen, besteht die möglichkeit ihnen über verschiedene Wege Daten zu übermitteln, die die Geräte zu einem fehlerhaften Verhalten veranlassen. Typischerweise enthalten die Daten, die dem Angreifer einen Zugang zu einem mobilen Endgerät ermöglichen, ausführbaren code. Häufig ist dieser Ausführbare Code in anderen Regulären Daten versteckt.

Durch die große Menge an Kommunikationsmöglichkeiten enthalten mobile Endgeräte viele Software Stacks, die den Protokollfluss regeln. Software Stacks wie Bluetooth und GSM wurden bereits in der Vergangenheit angegriffen [7, 8, 9].

Häufig wird Software von Desktop-PCs auf Mobilgeräte übertragen. Dabei müssen Anpassungen an das Programm vorgenommen werden um den eingeschränkten Resourcen auf den mobilen Geräten rechnung zu tragen. Die Softwareprodukte werden häufig parallel weiter entwickelt. Dadurch kann es dazu kommen, das ein Softwarebug in einer Desktopversion erkannt und behoben wurde. Währenddesses besteht der Softwarebug jedoch weiterhin auf dem Mobilgerät und ermöglicht so eine Ausnutzung der Schwachstelle.

Die von den Herstellern nachlässig eingepflegten Updates wurden von DeGusta [10] visualisiert. Sie zeigen, das ein Update zwar verfügbar ist, jedoch vom Hersteller des entsprechenden Geräts noch nicht eingepflegt wurde.

Die Applikationen auf den Geräten werden von tausenden von Entwicklern erstellt und über Webseiten der Betriebssystemhersteller verteilt. Um sicherzustellen das keine Malware über diese Kanäle aufgespielt wird, existieren verschiedene Sicherheitsmodelle [11]. Selbst in einemzentral überwachten software markt wie dem AppStore von Apple ist es Forschern gelungen malware zu demonstrationszwecken einzuschleusen [12].

Schadsoftware die Lücken ausnutzt für die es noch keine Viren-Signaturen gibt, nennt sich zero-day-exploits. Eine Erkennungssoftware sollte Schadsoftware auch ohne Signaturupdates erkennen können.

Eine Malware-Erkennungssoftware sollte auf Datenströmen operieren können, um Angriffe frühzeitig zu entdecken.

Um eine hohe Performance zu erreichen soll das Erkennungssystem den Kontext der empfangenen Daten kennen. Das System soll wissen ob ausführbarer Code in einem Datenstrom vorkommen darf.

3 Architektur und Technologie

Die eingesetzte Methode extrahiert im ersten Schritt eine statistische Funktion, die die Shannon-Entropie [13] nutzt. Anschliessend wird eine Fourier-Transformation durchgeführt um die Entropie-Funktion in den Frequenzbereich zu transformieren. Aus dem Frequenzbereich werden weitere statistische Daten extrahiert. Diese Daten werden im Erkennungsschritt an ein Neuronales Netz übergeben um eine binäre Klassifizierung in die Klassen *ausführbarer Code* und *kein ausführbarer Code* durchzuführen. Das Neuronale Netz wurde zuvor mit Testdaten aus ausführbarem und nicht ausführbarem Code trainiert.

Für die verschiedenen Verarbeitungsschritte gibt es Parameter die die Eigenschaften der Erkennung verändern können. Die Parameter des Verfahrens wurden systematisch getestet. Dabei wurde der Tradeoff zwischen Performance und Erkennungsgenauigkeit berücksichtigt um für mobile Endgeräte passende Einstellungen zu finden.

3.1 Extrahieren der Entropie-Funktion

Im ersten Verarbeitungsschritt nutzt das Verfahren die Shannon Entropie um eine Statistische Funktion aus den Rohdaten zu extrahieren. Im folgenden wird die *Shannon Entropie* mit *Entropie* abgekürzt.

Die Entropie ist ein maß für Ordnung in einer Zeichenfolge. Daher ist Sie eng Verwandt, wenn auch nicht Gleichbedeutend, mit der Komprimierbarkeit. Die Entropie kann mittels

$$H(X) = -\sum_{i=1}^{n} p(X_i) \log_2 p(X_i),$$

berechnet werden. Wobei X eine Symbolfolge aus einem endlichen Alphabet ist. Berechnet man die Entropie fensterweise auf einem Datenstrom, wie in Abbildung 1 dargestellt, ergibt sich eine Funktion des Datenstroms. Diese statistische Funktion kann bestimmte Eigenschaften der Rohdaten symbolisieren. Wird die Entropiefunktion beispielsweise für eine JPEG-Datei erzeugt dann lassen sich Bereiche mit niedriger Entropie am Anfang erkennen. Diese Bereiche mit niedriger Entropie enthalten bei JPEG-Dateien Header Informationen die unkomprimierten ASCII-Text enthalten. Im späteren Verlauf wird die Kurve ansteigen und somit den Anfang der stark komprimierten verlauf von Bilddaten anzeigen.

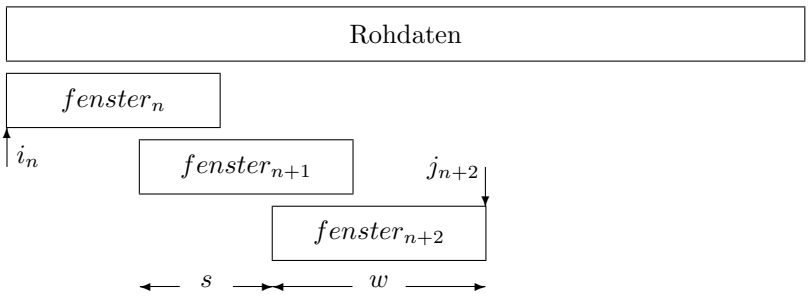


Figure 1: verallgemeinertes Fensterschema

3.2 Fourier-Transformation

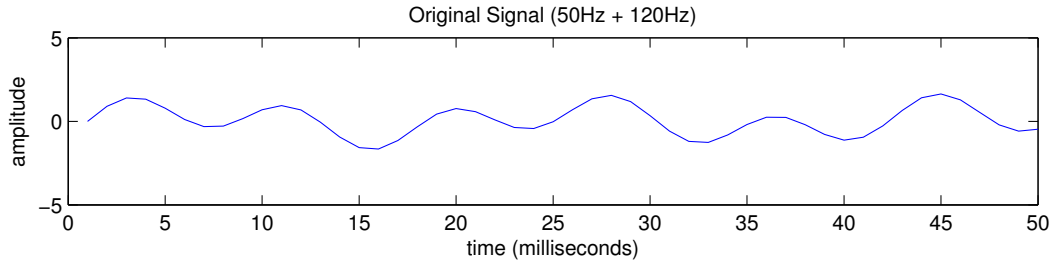
Mittels der Fourier-Transformation kann ein Signal vom Zeit in den Frequenzbereich transformiert werden. Dadurch können Eigenschaften eines Signals besser betrachtet werden, als im Zeitbereich. Die Betrachtung und interpretation der Fourier-Transformation nennt sich Fourier-Analyse. In Abbildung 2 wird ein Beispiel einer Fourier-Analyse gezeigt. In 2(a) wurden zwei Sinussignale mit unterschiedlicher Periodizität addiert. Im weiteren Schritt wurde in Abbildung 2(b) zufälliges Rauschen hinzugefügt. Die beiden Signale sind so nicht mehr erkennbar. Die Fourier-Transformation in 2(c) lässt mit zwei spitzen, die wesentlichen Eigenschaften des verrauschten Signals erkennbar werden. Die Fourier-Transformation ist im bereich der digitalen Signalverarbeitung weit verbreitet. Die Fourier-Transformation ist nur für statische Signale geeignet. Da die zu untersuchenden Signale nicht statischer-natur sind, nutzen wir die Kurzzeit-Fourier-Transformation (short-time Fourier transform (STFT))[14]. Diese kann mittels

$$\phi(t) = \int_{\infty}^{\infty} e^{ixt} f(x) dx$$

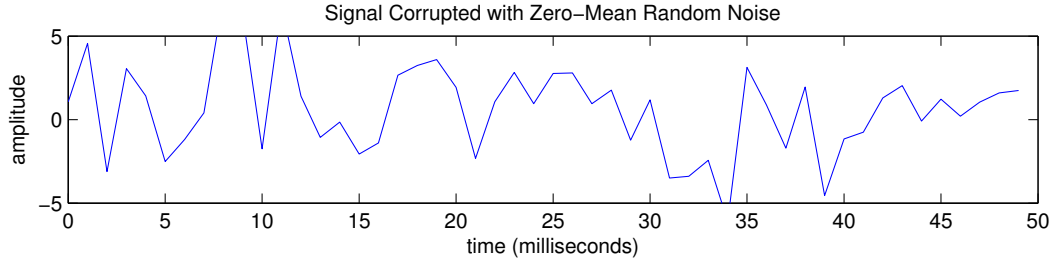
berechnet werden. Die STFT geht dabei nach demselben Fenster-Schema vor das in Abbildung 1 gezeigt wird.

3.3 Klassifikation und Erkennung

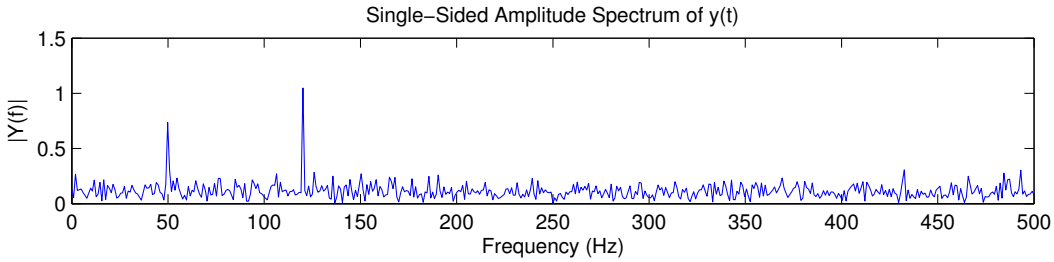
Im vorherigen Schritt wurde eine fensterweise Fourier-Transformation durchgeführt. In diesem Schritt wird eine Klassifikation in binary und non-binary datenblöcke durchgeführt.



(a) Zwei Sinussignale mit 50 Hz und 120 Hz.



(b) Abbildung 2(a) zufälligem Rauschen.



(c) Fourier-Transformation der Abbildung 2(b), zeigt spitzen bei 50 Hz und 120 Hz.

Figure 2: Beispiel einer Fourier-Analyse.

Um die Geschwindigkeit zu erhöhen und die Erkennungsrate zu verbessern, werden mehrere Entropie-Fenster blockweise an ein Neuronales Netz übergeben. Zuvor werden die erzeugten Fenster um weitere statistische Meßwerte ergänzt und anschließend einem Neuronalen Netz zur Klassifikation übergeben. Als statistische Meßwerte wird das arithmetische Mittel, der Median, sowie die absolute Abweichung vom Median berechnet.

3.4 Tests

In diesem Abschnitt werden verschiedene Parameter des Verfahres getestet. Zum testen des Verfahrens werden ELF-Arm-32 binaries verwendet, die die Binary-Klasse darstellen. Für die Klasse der non-binaries wurden häufig vorkommende Datentypen Verwendet, wie PDF,HTML und DOC-Dateien. Die getesteten Dateitypen sind in Spalte 1 der Tabelle 3 zu finden. Getestet wurde einem Trainingssatz und einem unabhängigem Testsatz an Daten. Es wurde mit 10MB pro Dateityp getestet.

3.4.1 Performance und Overhead

Um die Parameter zu testen ist es sinnvoll, den Bereich an Einstellungen einzugrenzen. Die Einstellungen der Parameter beeinflussen die Performance und die Erkennungsgenauigkeit des Systems. Es muss daher ein Kompromiß zwischen Erkennungsgenauigkeit und Performance gefunden werden. Die Tabelle 1 gibt die einstellbaren Parameter an.

Die offensichtlichsten Einschränkungen der Parameter sind: $w_e > o_e$ und $w_f > o_f$. Sie sind offensichtlich, weil die Überlappung mit dem vorherigen Fenster kleiner sein muss als das Fenster selbst, wie in Abbildung 1 zu sehen ist.

Der Overhead soll so gering wie möglich sein. Overhead definieren wir hier als den zusätzlichen Aufwand an Daten der im Vergleich zu den Eingabedaten (= 100%) anfällt. Mit $entropy_{overhead} = \frac{input_{size}}{w_e - o_e}$ berechnen wir den Overhead im Entropieschritt, beschrieben in Abschnitt 3.1. Mit $fourier_{overhead} = \frac{entropy_{overhead}}{w_f - o_f} \cdot w_f$ berechnen wir den Overhead im fourier-Schritt. Die größe des primitiven Datentyps double beträgt in Java 64-Bits (8 Byte).

parameter	description
w_e	Fenstergröße der entropie
o_e	Entropie: Überlappung mit dem Vorherigen Fenster
w_f	Fenstergröße der STFT
o_f	STFT: Überlappung mit dem Vorherigen Fenster
w_n	Anzahl der STFT-Fenster (entropie-spektren) die an das Neuronale Netz übergeben werden

Table 1: Parameter der Klassifizierung

w_e	o_e	w_f	o_f	w_n	overhead (MB)	minimale Verarbeitungsgröße (byte)
256	252	4	2	10	6	80
32	16	4	2	10	1.5	320
64	4	16	2	10	0.2856	8400

Table 2: Beispielparameter und ihre Auswirkungen auf das Gesamtsystem

Daher beträgt der gesamt-Overhead $(entropy_{overhead} + fourier_{overhead}) \cdot 8$. Die minimale Datenmenge die zur klassifizierung benötigt wird berechnet sich mittels $(w_e - o_e) \cdot (w_f - o_f) \cdot w_n$. Das bedeutet eine Datenmenge die unterhalb dieser Größe liegt kann von dem Verfahren nicht verarbeitet werden.

Die Tabelle 2 zeigt einige Beispiele für die Einstellungen der Parameter. Die Tabelle zeigt, das eine sehr hohe Überlappung zu einem sehr feinen Erkennungsbereich führt. Allerdings führt solch eine Einstellung auch zu sechs mal mehr Daten während der Verarbeitung. Die Parameter in Zeile 3 hingegen führen zu einem kleinen Overhead. Jedoch benötigt das Verfahren dann über 8000bytes bevor eine Erkennung stattfinden kann. Mit 8000 bytes würde eine Erkennung somit erst sehr spät beginnen.

3.4.2 Ergebnisse und Interpretation

Es wurden mehrere hundert Testläufe mit verschiedenen Parametern durchgeführt. In Tabelle 3 sind die Ergebnisse einer der Erfolgversprechenderen Testläufe gezeigt. Die Tabelle zeigt, das die falsch-Erkennungsrate bei PDF-Dateien noch recht hoch ist. Allerdings enthalten PDF-Dateien verschiedenste Datentypen in einem Containerformat. Daher stellen PDF-Dateien eine der problematischsten Dateitypen dar. Die Ergebnisse zeigen das eine Erkennung möglich ist, diese aber noch verbessert werden sollte.

4 Risiken

Die Praxistauglichkeit des Verfahrens ist nur gegeben, wenn eine sehr kleine falsch-positiv Rate erreicht werden kann. Ist die falsch-positiv Rate zu hoch, wird der Benutzer zu oft von dem Verfahren auf Schadcode hingewiesen, obwohl keiner an der betreffenden Stelle existiert. Selbst wenn die falsch-positiv Rate nicht weiter gesenkt werden kann, kann das Verfahren weiterhin als unterstützendes Mittel eingesetzt werden. Da das Verfahren sehr schnell arbeitet ist es denkbar einen zweistufigen Ansatz zu verwenden. Dabei wird im ersten Schritt durch dieses Verfahren geprüft ob Schadcode vorliegen könnte. Im zweiten Schritt könnte ein Aufwändigeres Verfahren prüfen ob wirklich Schadcode vorliegt.

Da es sich um ein statistisches Verfahren handelt, kann es nie eine hundert prozentige

Datei typ	% der Falsch Positiven	% der Falsch Negativen
doc	3,297%	0%
htm	0,41%	0%
pdf	7,8%	0%
ppt	7,82%	0%
xls	4,32%	0%
text	0,03%	0%
JavaScript	0,82%	0%
JPEG	6,93%	0%
Elf-arm-32	0	2,87%

Table 3: Klassifikations Ergebnisse eines 10 MB Test-datensatzes mit $w_e = 64$, $o_e = 16$, $w_f = 8$, $o_f = 2$ and $w_n = 1$

Erkennungsrate geben. Wird das Verfahren in der Praxis eingesetzt, ist abzusehen das Angreifer Gegenmaßnahmen ergreifen werden. Das verbreitete Framework Metasploit [1],zum testen von Exploits, bietet bereits mehrere möglichkeiten an, ausführbaren Code in ein Äquivalent zu transformieren, das schwerer Erkennbar ist. Einen weitergehenden Ansatz stellten Mason et al. **??** mit ihrem Paper zur transformation von ausführbarem Maschinencode in englishsprachige ASCII-Sätze vor. Die Autoren transformieren regulären X86 Maschinencode mit hilfe von äquivalenten Anweisungsblöcken in Maschinencode der ausschliesslich im ASCII-Bereich liegt. Durch ein eingesetztes Sprachmodell der englischen Sprache entstehen so vollständige, grammatikalisch korrekte Sätze die jedoch ausführbaren code darstellen. Die auf diese weise umcodierten Programme sind wesentlich größer und langsamer. Solche Ansätze sind mit statistische Verfahren, wie dem Vorgestellten, nur schwer erfassbar.

4.1 Aktueller Stand und Vorarbeiten

In Projekt 1 wurde ein Prototyp für das mobile Android-Betriebssystem entwickelt. Der Prototyp konnte zeigen das unser Verfahren funktioniert und mit entsprechender Performance auf mobilen Endgeräten lauffähig ist.

In Projekt 2 wird derzeit die Fehlererkennungsrate verschiedener Lernalgorithmen mit den in Projekt 1 verwendeten Neuronalen Netzen verglichen. Des weiteren soll die Fehlererkennungsrate durch optimierungen am Verfahren verbessert werden.

5 Zusammenfassung und Ausblick

In dieser Ausarbitung wurde ein Verfahren zur Erkennung von ausführbarem Code in beliebigen Datenströmen vorgestellt. Das vorgestllte Verfahren basiert auf statistischen Mustern in ausführbarem code. Daher benötigt es keine regelmäßigen Aktualisierungen, wie sonst bei traditionellen Virenscannern üblich.

Die bisherigen Experimente des Verfahrens haben gezeigt, dass das Verfahren wesentlich bessere Erkennungsrate liefern kann, als eine reine entropie-analyse. Bei einigen Einstellungen der Fenstergrößen des Verfahrens ergaben sich unerwartete Einbrüche bei der Erkennungsgenauigkeit. Im weiteren sollte untersucht werden ob die einbrüche bei der Erkennungsgenauigkeit an der Extraktionsphase oder in der Erkennungsphase liegen. Um bessere Erkenntnisse über die Erkennungsphase zu erlangen, sollen mehrere Algorithmen aus dem bereich Maschinenlernen auf das Erkennungsproblem angewandt werden. Dadurch soll erkennbar werden ob es für die Problemstellung bessere Methoden gibt als die bisher verwendeten Neuronalen Netze und deren eingesetzten Lernverfahren.

[1] `http://www.metasploit.com/`

References

[1] T. C. Schmidt, M. Wählisch, B. Jochheim, and M. Gröning, "WiSec 2011 Poster: Context-adaptive Entropy Analysis as a Lightweight Detector of Zero-day Shellcode Intrusion for Mobiles," *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R)*, vol. 15, no. 3, pp. 47–48, July 2011.

[2] AdaptiveMobile. (2011, February) Global security insight for mobile. [Online]. Available: http://www.adaptivemobile.com/global-security-insight-centre/mobile-report

[3] ECMA, *ECMA-340: Near Field Communication — Interface and Protocol (NFCIP-1)*. Ecma International, Dec. 2004. [Online]. Available: http://www.ecma.ch/ecma1/STAND/ecma-340.htm

[4] C. Mulliner, "Vulnerability Analysis and Attacks on NFC-enabled Mobile Phones," in *International Conference on Availability, Reliability and Security*, 2009. [Online]. Available: http://www.mulliner.org/nfc/

[5] T. C. Group. (2007, Aug.) Tcg specification architecture overview, revision 1.4. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14

[6] ——. (2007, June) Tcg mpwg mobile trusted module specification, version 1.0„ revision 1. [Online]. Available: https://members.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-trusted-module-1.0.pdf

[7] A. Moreno and E. Okamoto, "Bluesnarf revisited: Obex ftp service directory traversal," in *Proceedings of the IFIP TC 6th international conference on Networking*, ser. NETWORKING'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 155–166. [Online]. Available: http://dl.acm.org/citation.cfm?id=2039912.2039931

[8] D. Spill and A. Bittau, "Bluesniff: Eve meets alice and bluetooth," in *Proceedings of the first USENIX workshop on Offensive Technologies*, ser. WOOT '07. Berkeley, CA, USA: USENIX Association, 2007, pp. 5:1–5:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1323276.1323281

[9] C. Mulliner, N. Golde, and J.-P. Seifert, "SMS of Death: From Analyzing to Attacking Mobile Phones on a Large Scale," in *Proceedings of the 20th USENIX Security Symposium*, San Francisco, CA, USA, August 2011.

[10] M. DeGusta. (2011, October) Android orphans: Visualizing a sad history of support. [Online]. Available: http://theunderstatement.com/post/11982112928/android-orphans-visualizing-a-sad-history-of-support?45bebcc0

[11] C. Miller, "Mobile attacks and defense," *IEEE Security and Privacy*, vol. 9, no. 4, pp. 68–70, Jul. 2011. [Online]. Available: http://dx.doi.org/10.1109/MSP.2011.85

[12] A. Greenberg. (2011, July) iphone security bug lets innocent-looking apps go bad. [Online]. Available: http://www.forbes.com/sites/andygreenberg/2011/11/07/iphone-security-bug-lets-innocent-looking-apps-go-bad/

[13] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July/Oct. 1948.

[14] E. Jacobsen and R. Lyons, "The sliding dft," *Signal Processing Magazine, IEEE*, vol. 20, no. 2, pp. 74–80, March 2003.